3. Linguagem de Programação C

Prof. Renato Tinós

Departamento de Computação e Matemática
(FFCLRP/USP)

Principais Tópicos

3. Linguagem de programação C

3.1. Conceitos básicos

3.1.1. Introdução às Linguagens de Programação

3.1.2. Introdução à Linguagem C

3.1.2.1. Criação de programas em C

3.1.2.2. Erros de programação em C

3.1.1. Introdução às Linguagens de Programação

- **No início, os programas de computador eram escritos em linguagem de máquina**
  - Instruções primitivas que podiam ser executadas diretamente na máquina
    - » Ex: *001 10110*
  - Estrutura da linguagem refletia a estrutura da máquina
    - » Não reflete as necessidades do programador
    - » Difícil de programar

3.1.1. Introdução às Linguagens de Programação

- **Maior parte da computação envolvia o cálculo de fórmulas**
  - Fórmulas eram traduzidas para a linguagem de máquina
  - Por que não escrever programas parecidos com as fórmulas que se deseja computar?

3.1.1. Introdução às Linguagens de Programação

- **FORTRAN**
  - **FOR**MULA **TRAN**SLATION
  - Criado em 1950 por um grupo de programadores da IBM
  - Primeira linguagem de alto nível

3.1.1. Introdução às Linguagens de Programação

- **Linguagem de Baixo Nível**
  - As Instruções refletem diretamente o código de máquina
    - » Dependente da máquina utilizada
  - Exemplos: Assembler ou Assembly

- **Linguagens de Alto Nível**
  - Instruções genéricas que não precisam refletir diretamente o código de máquina
    - » Independem da máquina utilizada

3.1.1. Introdução às Linguagens de Programação

- **Linguagens de Alto Nível podem ser de vários tipos:**
  - Linguagens Não-Estruturadas
    - » Não usam chamadas de funções e procedimentos
    - » Exemplos: Cobol, Basic
  - Linguagens Procedurais
    - » Os programas determinam uma seqüência de chamadas de procedimentos
    - » Exemplos: C, Pascal
  - Linguagens Funcionais
    - » Utilizam expressões formadas por funções que procuram combinar padrões básicos
    - » Exemplos: Lisp, Prolog

3.1.1. Introdução às Linguagens de Programação

- **Linguagens de Alto Nível pode ser de vários tipos:**
  - Linguagens Orientadas a Objeto
    - » Utilizam procedimentos encapsulados (objetos) para representar as estruturas de dados
    - » Exemplos: Java, C++
  - Linguagens Visuais
    - » Utilizam ferramentas visuais de suporte para a programação
    - » Exemplos: Visual Basic, Delphi
  - Linguagens Específicas
    - » Linguagens para Bancos de Dados
      - Exemplos: Clipper, SQL
    - » Linguagens para Scripts
      - Exemplos: HTML, PostScript, PDF (*Portable Data Format*)

3.1.2. Introdução à Linguagem C

- **A origem da linguagem C**
  - Linguagem BCPL (Basic Combined Programming Language)
  - Desenvolvida em 1967
  - BCPL foi refinada para uma linguagem chamada B
  - Em 1972, Dennis Richie aperfeiçoou a linguagem B, para formar a linguagem C tradicional
    - » No início, C ficou conhecida como a linguagem de desenvolvimento do sistema operacional Unix
  - A rápida expansão de C para vários computadores levou a um grande número de variações na linguagem original
    - » Na década de 1980 foi criada a padronização ANSI C
      - Não ambígua
      - Independente de máquina

3.1.2. Introdução à Linguagem C

- **Principais vantagens da linguagem C**
  - Pode ser utilizada (adequada) para escrever programas que funcionam eficientemente
    - » Vários softwares básicos, sistemas de tempo real e programas de computação gráfica são escritos em C e seus derivados
      - Velocidade de execução é crítica nestas áreas
  - Portabilidade
    - » Mesmo programa C deve poder ser executado em máquinas e sistemas operacionais diferentes
      - Os próprios programadores C demandam consistência nas suas diferentes implementações

3.1.2. Introdução à Linguagem C

- **Principais vantagens da linguagem C**
  - Concisão
    - » Conjunto de operadores de C é muito poderoso
      - Várias operações podem ser combinadas em um único comando
    - » Gera códigos menores e mais elegantes
    - » Aumenta a produtividade dos programadores
      - Mais programas em menos tempo
  - Expansão de uso
    - » Surgimento de novas técnicas
    - » Crescimento das demandas por programas mais sofisticados

3.1.2. Introdução à Linguagem C

- **Principais vantagens da linguagem C**
  - Modularidade
    - » C incentiva a decomposição de um programa em módulos
    - » Diferentes partes do código podem ser escritos por programadores diferentes
    - » Torna a manutenção do código mais simples
    - » Facilita re-uso
      - Códigos reutilizados correspondem a cerca de 93,86% dos novos programas

3.1.2. Introdução à Linguagem C

- **No entanto, C tem suas deficiências**
  - Sua sintaxe não é clara no início
  - Sua flexibilidade pode levar o programador a erros
  - Programador, muitas vezes, deve ser criativo para propor as soluções de um problema

3.1.2.1. Criação de Programas em C

- **Sistema C**
  - Geralmente consiste de três partes
    - » O ambiente
    - » A linguagem
    - » A biblioteca padrão C

3.1.2.1. Criação de Programas em C

Processo de Compilação

Bibliotecas

```
0010100101010101010101001
0100110100101010101010101
1010101101001010101010111
1101010101001010101010101
...
```

Código Fonte

```
/* Programa na linguagem C       */

#include <stdio.h>

main( )
{
    printf ("Ola!!! \n");
}
```

link editor

Arquivo Executável

```
0010100101010101010101001
0100110100101010101010101
1010101101001010101010111
1101010101001010101010101
...
```

compilador

Código Objeto

```
0010100101010101010101001
0100110100101010101010101
1010101101001010101010111
1101010101001010101010101
...
```

3.1.2.1. Criação de Programas em C

- **Passos para a criação de um arquivo executável**
  - Edição
  - Compilação
    - » Pré-processamento
    - » Compilação
    - » Link-edição
  - Execução

3.1.2.1. Criação de Programas em C

- **Edição**
  - Processo de criação do código fonte através de um editor de textos
  - Programa é criado no editor e armazenado no disco
  - O arquivo deve ter a extensão **.c**
    - » Exemplo: prog_1.c
    - » Deve-se escolher um nome representativo para o arquivo
  - Os arquivos em C++ usam, em geral, a extensão .cpp

3.1.2.1. Criação de Programas em C

- **Compilação**
  - Processo de criação do código objeto a partir do código fonte
  - O programador executa um comando para criar o código objeto
  - Geralmente, se não houverem erros de programação, o compilador gera um arquivo com extensão .obj
  - Programa correspondente com instruções em linguagem de máquina

3.1.2.1. Criação de Programas em C

- **Pré-processamento**

  - Compiladores da linguagem C possuem um pré-processador
  - Antes de um programa C ser compilado, ele é pré-processado
    - » Linhas que começam com # se comunicam com o pré-processador
      - Exemplo: *#include <stdio.h>* faz com que o pré-processador inclua uma cópia do arquivo
        - Arquivo *stdio.h* é fornecido pelo sistema C

3.1.2.1. Criação de Programas em C

- **Link-edição**

  – Arquivo objeto é combinado com outros arquivos para produzir o arquivo executável

    » Outros arquivos objetos são arquivos pré-definidos em bibliotecas de funções (*libraries*)

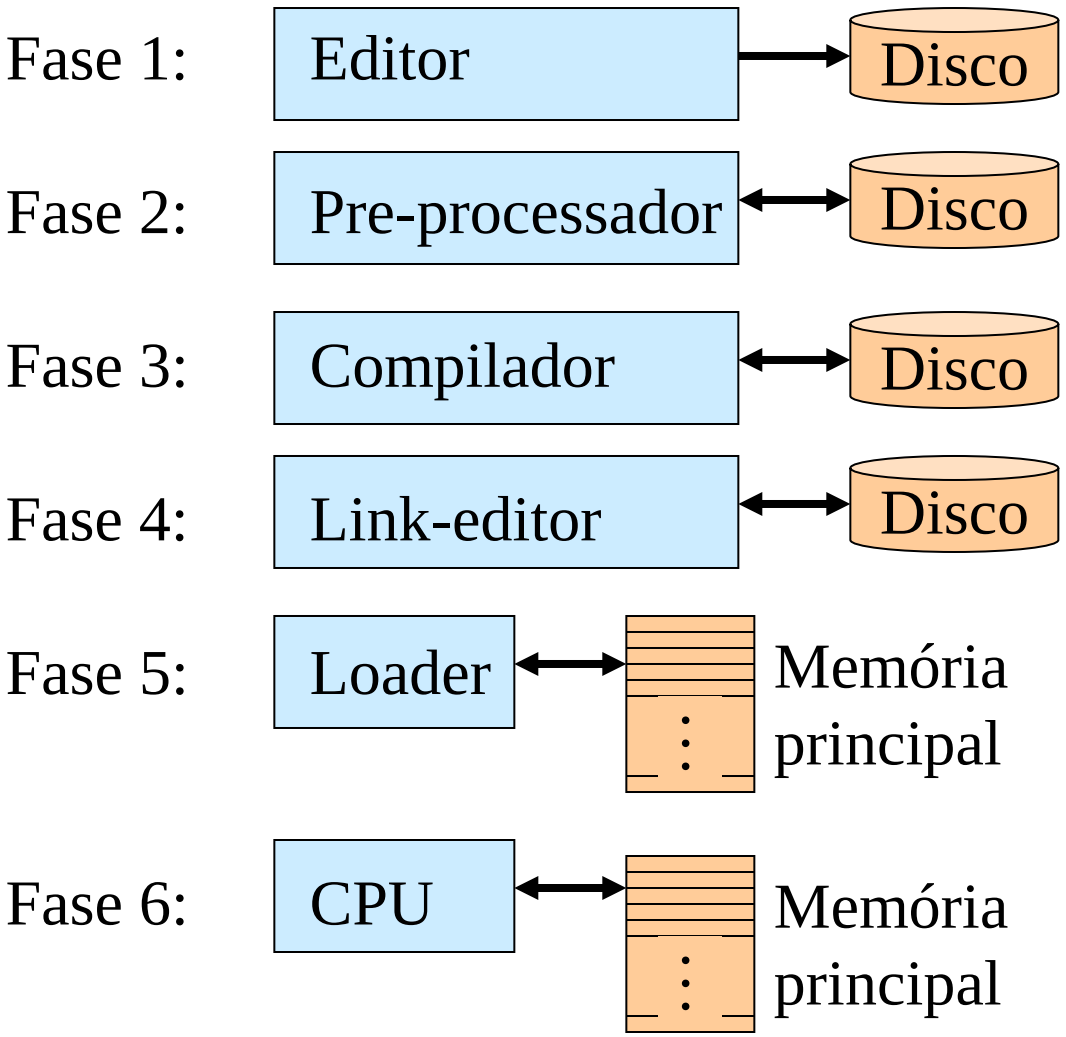
      • Contém instruções em linguagem de máquina para várias operações geralmente utilizadas pelos programas

    » O processo de combinação dos arquivos objetos em um arquivo executável é chamado de ligação

3.1.2.1. Criação de Programas em C

- **Execução**
  - Processo no qual o programa é executado
  - Carregador coloca o programa na memória principal
  - CPU executa cada instrução, possivelmente armazenando novos valores que o programa gera

3.1.2.1. Criação de Programas em C



Programa é criado em editor e armazenado em disco

Programa pré-processador processa o código

Compilador cria o código objeto e o armazena em disco

Link-editor associa o código objeto com as bibliotecas e armazena o novo código no disco

Coloca programa na memória

Executa cada instrução e, possivelmente, armazena novos valores

3.1.2.1. Criação de Programas em C

```
/* Programa ex_1.c       */

#include <stdio.h>

main( )
{
    printf ("Ola!!! \n");
}
```

Comentários do programa

Inclusão de bibliotecas

Programa principal

3.1.2.1. Criação de Programas em C

- **Padrões de programação**
  - Nomes de variáveis com significado
  - Código estruturado
  - Código adequadamente tabulado
  - Boa documentação
    - » Nome do programador e meio de contato
    - » Descrição geral
    - » Bons comentários

3.1.2.1. Criação de Programas em C

```
/* Programa  ex_1.c       */

#include <stdio.h>

main ( ) {
     int n, i, soma;
     float final;

     n = 4;
    soma = 0;
     for (i = 1; i < n; i++){
          soma = soma + i;
     }
     final = soma/(n-1);
     printf (“Resultado = %f”, final);
}
```

3.1.2.2. Erros de Programação

- **Muitas vezes, os programadores cometem erros ao escrever seus programas**

- **Em geral, um programa pode apresentar 4 categorias de erros**
  - Erros sintáticos ou de compilação
  - Erros de composição
  - Erros de execução
  - Erros semânticos

3.1.2.2. Erros de Programação

- **Erros sintáticos ou de compilação**
  - Código não obedece às regras de sintaxe da linguagem – não compila
    - » Sintaxe de uma linguagem é a sua “gramática”
  - Erros são indicados através de mensagens de erros resumidas na tela do computador
    - » Mensagens de erros mais detalhadas podem ser encontradas em manuais da linguagem
    - » Feitas as correções necessárias, a compilação pode ser reinicializada

3.1.2.2. Erros de Programação

```
/* Programa ex_2.c       */

#include <stdio.h>

main ( ) {
     int n, i, soma;
     float final;

     n = 4
     soma = 0;
     for (i = 1; i < n; i++){
          soma = sa + i;
     }
     final = soma/(n-1);
     printf ("Resultado = %f \n", final);
```

Ausência de ";"

Identificador Desconhecido

Falta final do Programa "}"

3.1.2.2. Erros de Programação

- **Erros de composição**
  - Ocorrem na fase de link-edição
  - Geralmente estão relacionados a erros na chamada de funções em outras bibliotecas
    - » Exemplo: chamar a função de saída de *prtl* ao invés de *printf*
    - » O link-editor não encontra a função *prtl* nas bibliotecas usadas
  - Erros são indicados através de mensagens de erros resumidas na tela do computador

3.1.2.2. Erros de Programação

```
/* Programa ex_3.c       */

#include <stdio.h>

main ( ) {
     int n, i, soma;
     float final;

     n = 4;
    soma = 0;
     for (i = 1; i < n; i++){
          soma = soma + i;
     }
     final = soma/(n-1);
     prin ("Resultado = %f", final);
}
```

Nome de função que não existe nas bibliotecas usadas

3.1.2.2. Erros de Programação

- **Erros de execução**
  - Código obedece às regras de linguagem
    - » programa é compilado
  - Mas ao ser executado, código executa operações não permitidas
    - » Exemplos
      - dividir por zero
      - raiz quadrado de número negativo
  - São mais difíceis de descobrir e interpretar

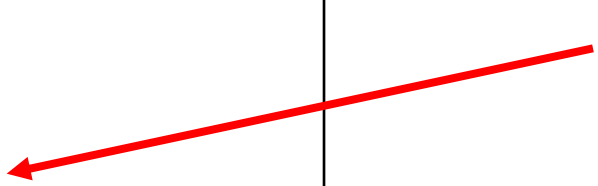
3.1.2.2. Erros de Programação

```
/* Programa ex_4.c       */

#include <stdio.h>

main ( ) {
     int n, i, valor, soma;
     float final;

     soma = 0;
     n = 4;
     for (i = 1; i <n; i++){
          soma = soma + i;
     }
     final = soma/(n-4);
     printf ("Resultado = %f", final);
}
```



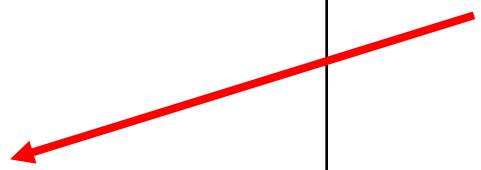
3.1.2.2. Erros de Programação

```
/* Programa ex_2.c       */

#include <stdio.h>

main ( ) {
    int n, i, valor, soma;
    float final;

    soma = 0;
    n = 4;
    for (i = 1; i >0; i++){
        soma = soma + i;
    }
    final = soma/(n-1);
    printf ("Resultado = %f", final);
}
```



Programa nunca termina execução

3.1.2.2. Erros de Programação

- **Erros semânticos**
  - Semântica = significado
  - Erros no projeto lógico do programa
    - » Forma utilizada para resolver o problema
  - Provoca erros indesejáveis no programa
    - » Mensagens indicativas dos erros não são apresentadas
    - » Sua correção exige mudanças na concepção do programa
  - São os erros de detecção mais difícil

3.1.2.2. Erros de Programação

```
/* Programa ex_3.c       */

#include <stdio.h>

main ( ) {
     int n, i, soma;
     float final;

     n = 4;
    soma = 0;
     for (i = 1; i < n; i++){
              soma = i;
     }
     final = soma/(n-1);
     printf ("Resultado = %f", final);
}
```

Observe que o resultado esperado (calcular a média dos *n-1* números) não é produzido!!!